Wrocław, 12 maja 2024

Paradygmaty Programowania Obiektowego

Laboratorium 4

Szablony klas i funkcji

prowadzący: *mgr inż. Marta Lampasiak*

1 Wprowadzenie

Szablony pozwalają na tzw. **programowanie uogólnione**, czyli pisanie fragmentów kodu, które następnie mogą zostać użyte dla danych różnych typów. Oczywiście, podczas tworzenia danego fragmentu programista musi przyjąć pewne założenia co do operacji możliwych do wykonania na danych podlegających przetwarzaniu. Szablony można tworzyć dla funkcji oraz dla klas.

Szablony funkcji

Szablon funkcji jest mechanizmem do tworzenia zestawu bardzo podobnych funkcji – identycznych w działaniu, ale różniących się jedynie typem obiektów, na których pracują. Sygnałem dla kompilatora, by wygenerował daną funkcję szablonową, jest miejsce w programie, gdzie taką funkcję wywołujemy lub gdy chcemy skorzystać z jej adresu.

Szablon funkcji zaczyna się od słowa kluczowego `template`, po którym następuje parametr (lub parametry) szablonu umieszczony wewnątrz $< >$, a następnie poniżej znajduje się definicja funkcji. W powyższym kodzie `T` jest argumentem szablonu, który akceptuje różne typy danych (np. int, float, double itp.), a `typename` jest wspomnianym słowem kluczowym.

```
template <typename T>
T functionName(T parameter1, T parameter2, ...)
{
    // code - function body
}
```

Gdy argument typu danych jest przekazywany do funkcji `functionName()`, kompilator generuje nową wersję funkcji `functionName()` dla danego typu danych.

Zadeklarowany szablon funkcji można wywołać w innych funkcjach lub szablonach, jak na przykład w funkcji `main()` w następujący sposób:

```
functionName<dataType>(parameter1, parameter2,...);

// for example - calling with int parameters
result_Function = functionName<int>(2, 3);
```

Szablony klas

Szablon klasy tak samo jak szablon funkcji zaczyna się od słówka kluczowego `template`, następnie występują parametry zawarte w $< >$, później pod spodem umieszcza się deklarację klasy.

```
template <class T>
class className {
  private:
    T var;
    ... .. ...
  public:
    T functionName(T arg);
    ... .. ...
};
```

W powyższej deklaracji `T` jest argumentem szablonu, który jest symbolem zastępczym dla używanego typu danych, a `class` jest słowem kluczowym. Wewnątrz ciała klasy zmienna składowa `var` i funkcja składowa `functionName()` są typu `T`.

Poniżej znajduje się przykład utworzenia obiektu szablonu klasy.

```
className<dataType> classObject;

// for example
className<int> Object1;
className<float> Object2;
className<string> Object3;
```

Szablony klas a podział na pliki

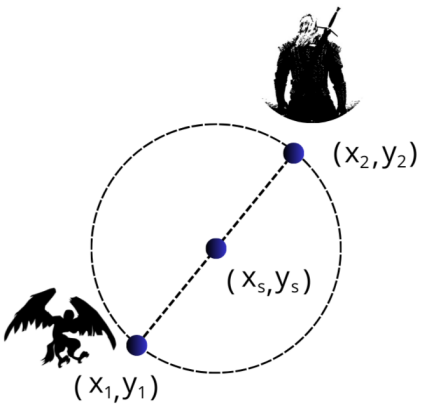
W przypadku szablonu klas pojawia się pytanie w jaki sposób rozmieścić je w plikach. Definicję szablonu klas z deklaracjami umieszczamy w **pliku nagłówkowym**. Tak samo jak dla „zwykłych" klas, w ciele szablonu klas mogą zostać umieszczone nie tylko deklaracje metod, ale także ich definicje. Zwykle postępuje się tak, kiedy metody są „krótkie" i „proste". W przypadku większych funkcji składowych, których ciała są na tyle „duże", że zostają zapisane poza ciałem szablonu zachodzi pewna różnica w zapisie w porównaniu do postępowania dla „zwykłych" klas. Jednym z podejść jest umieszczenie definicji funkcji składowych szablonu klasy również w pliku nagłówkowym, ale poniżej, już po zakończeniu definicji ciała szablonu klas. Następnie plik nagłówkowy zostaje załączony do każdego pliku programu, który korzysta z danego szablonu.

Gdzie już miałeś do czynienia z szablonami

Warto wspomnieć, że znane powszechnie kontenery biblioteczne są przykładami programowania ogólnego i zostały opracowane przy użyciu koncepcji szablonów. Istnieje jedna definicja każdego kontenera, na przykład *vector*, ale możemy zdefiniować wiele różnych rodzajów wektorów, na przykład $vector< int >$ lub $vector< string >$.

2 Zadanie

W grze, którą tworzysz potrzebne jest Ci narzędzie, które pozwoli Ci określić zależności położenia między jej jednostkami: postaciami i wrogami. Jest to dla Ciebie szczególnie ważne w momencie, kiedy np. Twoja postać musi stoczyć walkę z wrogiem lub zostaje osaczona przez większą ilość potworów. Przydatne mogą okazać się takie informacje jak: odległość między nimi lub chociażby obwód areny, która określana jest jako okrąg o środku w punkcie (x_s, y_s) leżącym w połowie odległości między nimi. Poniżej znajdziesz rysunek poglądowy (rys. 1), który pozwoli Ci na zrozumienie opisanej sytuacji.



Rysunek 1: Schemat poglądowy zależności położenia

W związku z tym, że współrzędne punktu mogą zostać wyrażone zarówno np. poprzez wartości całkowite, jak i zmiennoprzecinkowe – **należy wykorzystać szablony klas**. Współrzędne zapiszesz więc dla jakiegoś ogólnego typu, np. `T`. Natomiast w ogólności może być to każda, inna litera, `T` jest najpopularniejsza od słowa `template`. Pamiętaj więc pisząc program, że w wielu miejscach zmienne będą „typu T", aby potem móc w ich miejsce podstawić np. typ `int`.

Ogólna struktura programu

1. Szablon klasy reprezentującej punkt – klasa `Point`
2. Szablon klasy implementującej porównanie – klasa `Comparison`.
3. Szablony funkcji służące do generacji losowego punktu.

Wymagania i podpowiedzi

1. Podziel odpowiednio kod na pliki źródłowe i nagłówkowe, korzystając z informacji zawartych we wstępie teoretycznym i z wiedzy z poprzednich zajęć (na przykład dotyczącej nazewnictwa plików),
2. Definicje implementowanych metod umieść poza klasami (w klasie może znaleźć się jedynie definicja konstruktora, destruktora, *gettery* oraz *settery*, ale jeżeli chcesz mogą być też poza klasą).
3. Jeżeli dla metod lub pól składowych szablonu klasy nie podano inaczej, zastosuj etykietę dostępową *public*.
4. Pamiętaj, aby zmienne nazywać względem tego, co wyrażają – nie stosuj skrótów, które mogą być niezrozumiałe dla odbiorcy oraz do pól klasy dodaj: `_` („podłogę"). Pisz tak, aby kod „sam się komentował".
5. Jeżeli chcesz, niech destruktor dodatkowo wyświetla tekst informujący o usunięciu obiektu.
6. Jeżeli uznacz, że potrzebny jest Ci inny konstruktor, bądź jakaś inna funkcjonalność i ma ona uzasadnienie w kontekście tworzonego programu – dodaj ją, możesz również ją ze mną skonsultować podczas zajęć.
7. Nie pisz programu od początku do końca bez sprawdzania jego działania w trakcie! Jeżeli zaimplementujesz pewną część, którą możesz przetestować – zrób to!
8. **Podpowiedź**: W celu lepszego zrozumienia zadania i podsumowania tego, co należy wykonać jako zadanie możesz spróbować rozrysować sobie diagram klas UML (podobny do tego, który został zamieszczony w instrukcji do poprzedniego laboratorium). Jeżeli nie występuje dziedziczenie, ale występuje pewna zależność, oznacz to strzałką narysowaną przerywaną linią. Nie jest to zadanie obowiązkowe, ale myślę, że może ułatwić Ci pracę podczas zajęć i zebrać wymagania w jednym miejscu. Zamiast schematu UML możesz też po prostu pomyśleć o tym, jak o narysowaniu schematu swojego programu, abyś wiedział, jakie kroki chcesz wykonać.

Opis szczegółowy

Szablon klasy `Point` zawiera:

- Dwa pola składowe reprezentujące współrzędne punktu, które są *private*.
- Konstruktor wieloargumentowy (użyj listy inicjalizacyjnej),
- Destruktor.
- Metody typu `get` dla pól klasy (*gettery*).

- Metody typu `set` dla pól klasy (*settery*).
- Metodę wyświetlającą współrzędne punktu.

Szablon klasy `Comparison` zawiera:

- Trzy pola składowe o etykiecie *private*: dwa punkty (obiekty klasy `Point`) oraz pole `solution_`.
- Konstruktor wieloargumentowy (użyj listy inicjalizacyjnej).
- Destruktor.
- Potrzebne Ci metody `get` oraz `set`.
- Metodę:
  1. wyświetlającą współrzędne obydwóch punktów,
  2. obliczającą odległość pomiędzy dwoma punktami, która nie zwraca jej, a zapisuje wynik w polu klasy `solution_`.Tak więc według rys. 1 odległość pomiędzy (x_1, y_1) oraz (x_2, y_2).
  3. obliczającą tzw. punkt środkowy, która zwraca punkt z nowymi współrzędnymi (współrzędne punktu (x_s, y_s), czyli punktu dokładnie pomiędzy np. wrogiem a Twoją postacią – jednocześnie jest to środek okręgu, na którym oby dwoje się znajdują,
  4. obliczającą (zwracającą) obwód okręgu, na którym znajdują się jednostki w grze,
  5. wyświetlającą wszystkie informacje możliwe do uzyskania z porównania punktów (ich współrzędne, odległość, punkt środkowy, obwód).

Szablony dwóch metod do generowania współrzędnych punktów:

- Nazwane dokładnie tak samo, ale przyjmujące inne argumenty – uzyskaj przesłanianie się szablonów funkcji
- Pierwsza metoda przyjmuje punkt, którego współrzędne mają zostać wylosowane. Skorzystaj z odpowiedniej funkcji bibliotecznej. Niech metoda zapewnia, aby losowane współrzędne były za każdym razem inne. Metoda nie przyjmuje żadnego zakresu losowanego punktu, pozostaw zakres domyślny.
- Druga metoda jako argumenty, oprócz punktu, którego współrzędne zostaną wylosowane, przyjmuje również dwa parametry: `T start` oraz `T end`, które reprezentują zakres z jakiego ma zostać wylosowana współrzędna. Niech metoda zapewnia, aby losowane współrzędne były za każdym razem inne.

Ocenianie

Polecenia *ocena 3.0*

1. Implementacja szablonu klasy `Point`.
2. Implementacja szablonu klasy `Comparison` z dwoma pierwszymi metodami (wyświetlającą współrzędne obydwóch punktów, obliczającą odległość).
3. Podział na pliki nagłówkowe i źródłowe.
4. Zaprezentowanie działania programu (zaimplementowanych metod) dla zmiennych typu `int` oraz `double` (czyli sprawdzenie działania szablonu).

Polecenia *ocena 4.0*

Wykonanie zadań na ocenę 3.0 i napisanie (oraz zademonstrowanie działania) wszystkich podanych metod.

Polecenia *ocena 5.0*

Wykonanie zadań na ocenę 4.0 oraz:

- zaimplementowanie i zademonstrowanie działania obu metod służących do generowania współrzędnych punktów,
- dopisanie jednej, wymyślonej funkcjonalności do klasy `Comparison`, która będzie według Ciebie pasować do mechaniki gry w kontekście położenia jej jednostek lub dopisanie jednego wymyślonego szablonu funkcji, który będzie szablonem wieloparametrowym (ang. *Function templates with multiple template type parameters*).

Podsumowanie – Interpretacja zadania

Zauważ, że jeżeli uda Ci się zaimplementować powyższe zależności matematyczne, to przykładowo: już wiadomo, jak daleko znajduje się wróg od Twojej postaci, jaki jest obwód areny, na której być może będą walczyć, znasz również miejsce równoodległe od Was oby dwojga. Matematycznie opisałeś mechanikę gry i uzyskałeś w jej kontekście wiele informacji.

Motywacja do pracy

„To, co znane, przestaje być koszmarem. To, z czym umie się walczyć, nie jest już tak groźne."

A.Sapkowski